Universität Leipzig
Institut für Informatik
Abteilung Datenbanken
Prof. Dr. E. Rahm
T. Böhme, R. Müller, Dr. P. Richter

Ausgabe: 08.04.2002
Abgabe: 22.04.2002

Algorithmen und Datenstrukturen 2
SS 2002 - Übungsblatt 1

Hinweis: Alle *Listings* müssen in der ersten Zeile nach der Klassendefinition den *Namen des Programmautors* als Kommentar enthalten. Ohne diesen Namen werden die Listings **nicht gewertet**. Ebenfalls führt das Fehlen eines geforderten Ausdrucks der Programmausführung zur Nichtbewertung der Aufgabe.

1. Aufgabe (Einfügen in AVL-Bäumen) (8 Punkte)

a) Ist der Aufbau eines AVL-Baumes unabhängig von der Einfügereihenfolge? Wenn ja, begründen Sie dies. Wenn nein, geben Sie ein Gegenbeispiel an.

Gegeben sei nun der folgende AVL-Baum mit Komponistennamen:

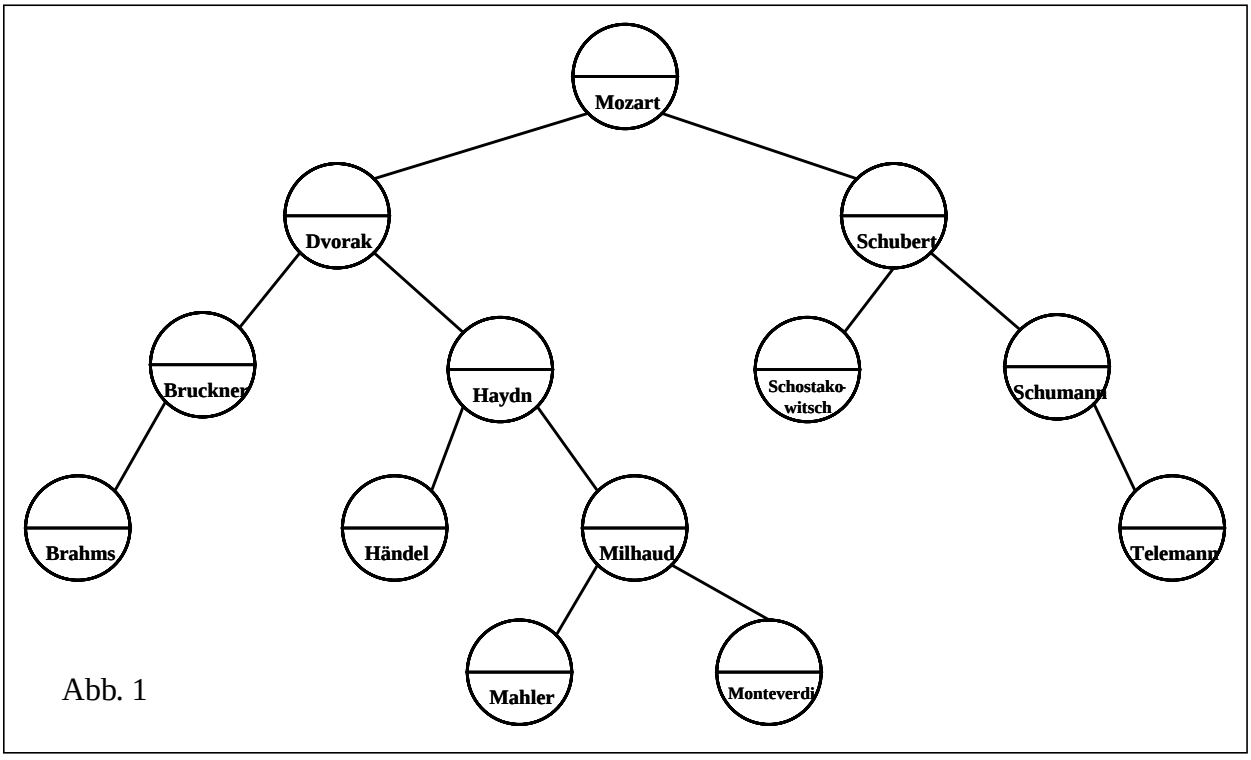

Abb. 1

b) Notieren Sie für jeden Knoten des obigen Baumes den Balancierungsfaktor.

c) Fügen Sie Knoten mit den Schlüsseln "Beethoven" und "Zelenka" ein. Aktualisieren Sie hierbei für *jeden* betroffenen Pfad die Balancierungsfaktoren, und führen Sie ggf. die nötigen Rebalancierungen durch, um die AVL-Eigenschaft zu erhalten, und geben Sie insbesondere die Rotationstypen an.

d) Fügen Sie einen Knoten mit dem Schlüssel "Mendelssohn" ein. Aktualisieren Sie auch hier die Balancierungsfaktoren und führen Sie die nötigen Rebalancierungen durch.

2. Aufgabe (Löschen in AVL-Bäumen) (4 Punkte)

Gegeben sei der folgende AVL-Baum:

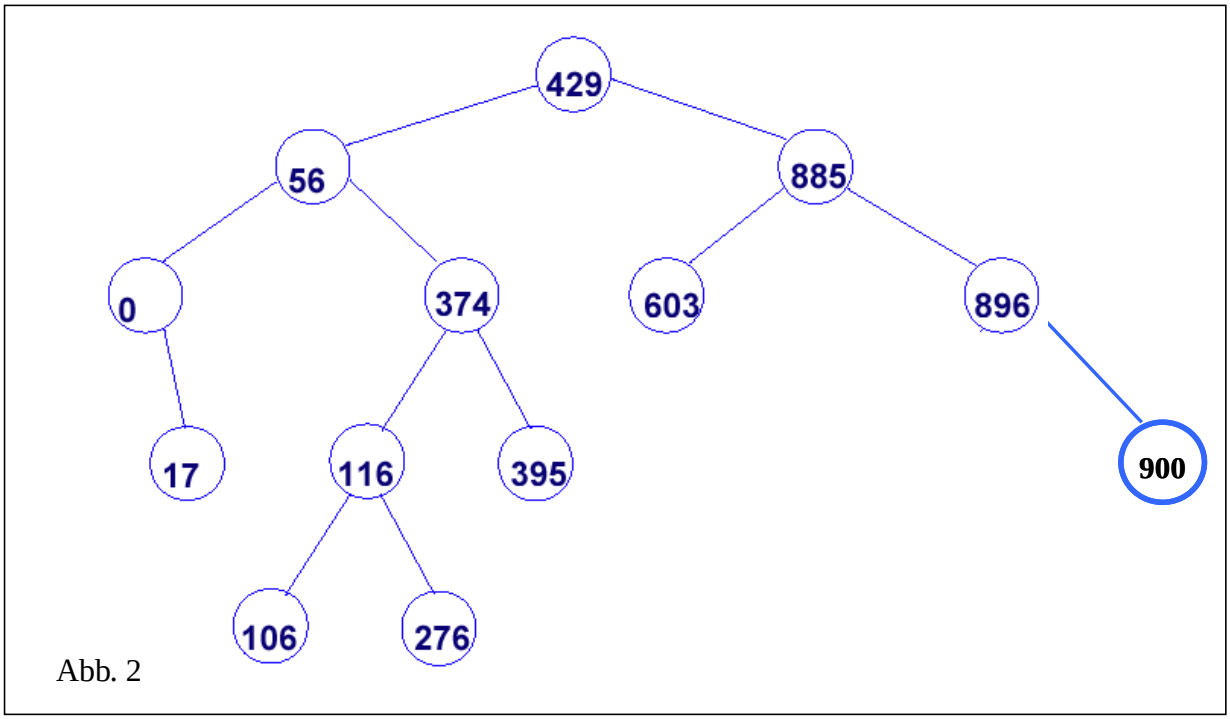

Abb. 2

Löschen Sie den Knoten 603, und führen Sie ggf. die nötigen Rotationen durch.

3. Aufgabe (Positionssuche mit balancierten Bäumen) (6 Punkte)

Betrachten Sie wiederum den in Abb. 1 gezeigten AVL-Baum.

a) Geben Sie zu jedem Knoten dieses Baumes den *Rang* an.

b) Bestimmen Sie auf der Basis der in Teilaufgabe a) ermittelten Rangzahlen und des in der Vorlesung vorgestellten Suchalgorithmus das 6. Element in diesem Baum. Notieren Sie jeden einzelnen Schritt bei Ihrer Suche, und geben Sie für jeden einzelnen Schritt die aktuellen Werte der Position *p* und des Ranges *r* an.

c) Geben Sie die mittleren Suchkosten für den in Teilaufgabe b) benutzten Algorithmus an.

d) Vergleichen Sie die Komplexität der Positionssuche mit balancierten Bäumen mit der Komplexität der in Kap. 2 (aus "Algorithmen und Datenstrukturen 1") vorgestellten Verfahren (Auswahlproblem).

4. Aufgabe (M-Wege-Suchbäume) (12 Punkte)

Implementieren Sie in Java das Einfügen von Schlüssel-Wert-Paaren in einen M-Wege-

Suchbaum. Auf der Web-Übungsseite haben wir Ihnen dafür die Klasse *MWaySearchTree* bereitgestellt, die einen M-Wege-Suchbaum teilweise implementiert. Die darin als Rumpf vorgegebene Methode *public void insert (Orderable key, Object obj)* ist von Ihnen zu implementieren. Sie soll einen neuen Schlüsselwert (key) mit dazugehörigem Datenobjekt (obj) in den Baum einfügen. Die Baumknoten werden durch die Klasse MNode repräsentiert (identisch zur Definition aus der Vorlesung auf Folie 1-15). Der Einfügealgorithmus soll gemäß dem in der Vorlesung (Folie 1-13) gezeigten Verfahren arbeiten. Soll ein Schlüssel eingefügt werden, der schon im Baum existiert, so wird nur das Datenobjekt aktualisiert. Die Einfügemethode muss die Baumhöhe, die in der Variablen *height* gespeichert ist, bei Bedarf anpassen.

Überprüfen Sie die Implementierung mit dem ebenfalls bereitgestellten Programm *Ueb1Test*. Für das Übersetzen und Ausführen des Programms benötigen Sie die auf der Web-Übungsseite bereitgestellten Dateien *MNode.java, MWaySearchTree.java, Orderable.java, OrderableInt.java, Ueb1Test.java*.

Abgabe:

1. dokumentiertes Listing des Quellprogramms (MWaySearchTree.java)
2. Ausgabe eines Programmlaufs von Ueb1Test